Sistemas de gestión de flujo de trabajo (workflow management systems - WFS)

María Ángeles Mendoza Pérez – Curso CSIC reproducibilidad 2022
27 de Abril de 2022

¿Qué es un flujo de trabajo - Workflow?

¿Qué es un flujo de trabajo - Workflow?

Intuitivamente: **secuencia de tareas** que fluirá de unas a otras hasta que el proceso se realice

¿Qué es un flujo de trabajo - Workflow?

Intuitivamente: secuencia de tareas que fluirá de unas a otras hasta que el proceso se realice

Hacer desayuno:

1. Cortar el pan
2. Colocarlo en la tostadora
3. Encender tostadora
4. Esperar que se tueste
5. Retirar el pan de la tostadora
6. Untar mantequilla al pan
7. ....

¿Qué es un flujo de trabajo - Workflow?

Intuitivamente: secuencia de tareas que fluirá de unas a otras hasta que el proceso se realice

Hacer desayuno:

1. Cortar el pan
2. Colocarlo en la tostadora
3. Encender tostadora
4. Esperar que se tueste
5. Retirar el pan de la tostadora
6. Untar mantequilla al pan
7. ....

Más formalmente:

- Lista de **tareas** o operaciones
- Conjunto de **dependencias** entre interconectadas tareas (el flujo)
- Conjunto de **recursos** (datos) necesarios para generar el flujo

¿Qué es un flujo de trabajo - Workflow?

Flujo de trabajo científico:

¿Qué es un flujo de trabajo - Workflow?

Flujo de trabajo científico: serie **repetible** de pasos que procesan datos u otro información

→ re-ejecución de WFs con **diferentes** parámetros y/o datasets

¿Qué es un flujo de trabajo - Workflow?

Flujo de trabajo científico: serie **repetible** de pasos que procesan datos u otro información

→ re-ejecución de WFs con **diferentes** parámetros y/o datasets

Tipos más comunes:

- WF de laboratorio
- WF de análisis de datos
- WF computacionales:

¿Qué es un flujo de trabajo - Workflow?

Flujo de trabajo científico: serie **repetible** de pasos que procesan datos u otro información

→ re-ejecución de WFs con **diferentes** parámetros y/o datasets

Tipos más comunes:

- WF de laboratorio
- WF de análisis de datos
- WF computacionales: conjunto de **pasos computacionales** y el **manejo de datos** necesarios diseñados para alcanzar un objetivo

Ciencia computacional →

Incrementado notablemente los descubrimientos experimentales y teóricos

- Gran cantidad de datos asequibles (almacenamiento y comunicación)
- Mayores recursos computacionales

Ciencia computacional →
Incrementado notablemente los descubrimientos experimentales y teóricos

- Gran cantidad de datos asequibles (almacenamiento y comunicación)
- Mayores recursos computacionales

Se caracterizan por procesar sistemáticamente **grandes volúmenes** y/o **complejas** colecciones de datos:

- **Mover datos** de las fuentes a los recursos computacionales (**distribuidos**)
- **Pre- procesamientos** de datos (limpiar, normalizar, etc.)
- **Computación intensiva** de software.
- **Conservar y distribuir** los resultados, meta-datos, documentación (**archivo**)

Flujos de trabajo y Ciencia Computacional

Retos:

- Escala, complejidad y heterogeneidad de datos
- Cantidad de computación: HPC, clusters, grids, clouds
- Actualización y Modificación → propagación a otros procesos dependientes
  - * Ordenada y metódicamente
  - * Mayor complejidad mayor probabilidad de errores

Flujos de trabajo y Ciencia Computacional

Retos:

- Escalar, complejidad y heterogeneidad de datos
- Cantidad de computación: HPC, clusters, grids, clouds
- Actualización y Modificación → propagación a otros procesos dependientes
  - * Ordenada y metódicamente
  - * Mayor complejidad mayor probabilidad de errores

Manual ejecución y mantenimiento de un WF es:

- Tediosa
- Susceptible a errores
- Inviable, mayoría ocasiones
- Pobremente documentado
- Se reinventa la rueda para común tareas

¿Qué son los sistemas de flujo de trabajo - WFS?

¿Qué son los sistemas de flujo de trabajo - WFS?

Herramientas que permiten **representar** y **automatizar** el proceso (flujo de trabajo)

→ **Fiable y Eficiente**

Soportan:
- Especificación
- Modificación
- Ejecución
- Depuración
- Reconfiguración
- Inteligente re-ejecuciones
- Monitoreo

¿Qué son los sistemas de flujo de trabajo - WFS?

Herramientas que permiten **representar** y **automatizar** el proceso (flujo de trabajo)
→ **Fiable y Eficiente**

Soportan:
- Especificación
- Modificación
- Ejecución
- Depuración
- Reconfiguración
- Inteligente re-ejecuciones
- Monitoreo

Popularidad como acercamiento orientado a usuario

- Proporciona un entorno fácil de usar
- Simplifica el proceso de compartir y reutilizar WF → experimento a gran escala
- Coordina y automatiza la provisión de recursos: locales y distribuidos y heterogéneas plataformas (GREEN COMPUTING)

Beneficios de los WFS

Beneficios de los WFS

FIABILIDAD

- Mantenimiento y actualización de los WF más sencillo
- Depuración. Permite corregir errores y mejorar resultados

Beneficios de los WFS

FIABILIDAD

- Mantenimiento y actualización de los WF más sencillo
- Depuración. Permite corregir errores y mejorar resultados

EFICIENCIA

- Tiempo de procesamiento: paralelización, cuellos de botella, reusar resultados
- Repetibilidad. datasets y/o parámetros. Inteligentes re-ejecuciones
- Re-usabilidad: fragmentos de WFs en otros WFs

Beneficios de los WFS

FIABILIDAD

- Mantenimiento y actualización de los WF más sencillo
- Depuración. Permite corregir errores y mejorar resultados

EFICIENCIA

- Tiempo de procesamiento: paralelización, cuellos de botella, reusar resultados
- Repetibilidad. datasets y/o parámetros. Inteligentes re-ejecuciones
- Re-usabilidad: fragmentos de WFs en otros WFs

COLABORACIÓN

- WF puede ser almacenado y compartido
- Sinergias multidisciplinares y experimentos computaciones a gran escala
- Portabilidad.
- Conocimiento basados en acumulativos WF o fragmentos

Beneficios de los WFS

FIABILIDAD

- Mantenimiento y actualización de los WF más sencillo
- Depuración. Permite corregir errores y mejorar resultados

EFICIENCIA

- Tiempo de procesamiento: paralelización, cuellos de botella, reusar resultados
- Repetibilidad. datasets y/o parámetros. Inteligentes re-ejecuciones
- Re-usabilidad: fragmentos de WFs en otros WFs

COLABORACIÓN

- WF puede ser almacenado y compartido
- Sinergias multidisciplinares y experimentos computaciones a gran escala
- Portabilidad.
- Conocimiento basados en acumulativos WF o fragmentos

VISIBILIDAD

Más producción científica

Sistemas de flujo de trabajo y Provenance

Reproducibilidad es una característica fundamental de las buenas prácticas científicas y el avance del conocimiento.

Se considera un componente emergente pero esencial de la **Ciencia Abierta** y para la instauración de los **FAIR principios**:

- Findabililty
- Accessibility
- Interoperability
- **Reusability**

Sistemas de flujo de trabajo y Provenance

Provenance. Tipo de meta-datos que registran la historia de obtención de los productos de datos y los pasos del WF → **Flujo pueda ser reproducidos**

Sistemas de flujo de trabajo y Provenance

Provenance. Tipo de meta-datos que registran la historia de obtención de los productos de datos y los pasos del WF → **Flujo pueda ser reproducidos**

- elementos de hardware y software
- dependencias
- configuración
- bases de datos y datasets
- datos intermedios

Sistemas de flujo de trabajo y Provenance

Provenance. Tipo de meta-datos que registran la historia de obtención de los productos de datos y los pasos del WF → **Flujo pueda ser reproducidos**

- elementos de hardware y software
- dependencias
- configuración
- bases de datos y datasets
- datos intermedios

Beneficios

- Análisis y depuración de resultados: comparaciones, explicaciones, detección errores
- Transparencia. Confianza en los resultados
- Ciencia Reproducible.
- Longevidad

Sin embargo, específicos de los WF y difícil de integral en diferentes sistemas

Características de WFS

Gran variedad de WFS, desde más alto a más bajo nivel

- Usuarios expertos o no. Interfaz de usuario
- Aprender un lenguaje de programación
- Independiente o no del código
- Representaciones conceptuales (abstracción) → No contiene información de acceso a los recursos
  * Tratar la complejidad
  * Portabilidad a través de entornos

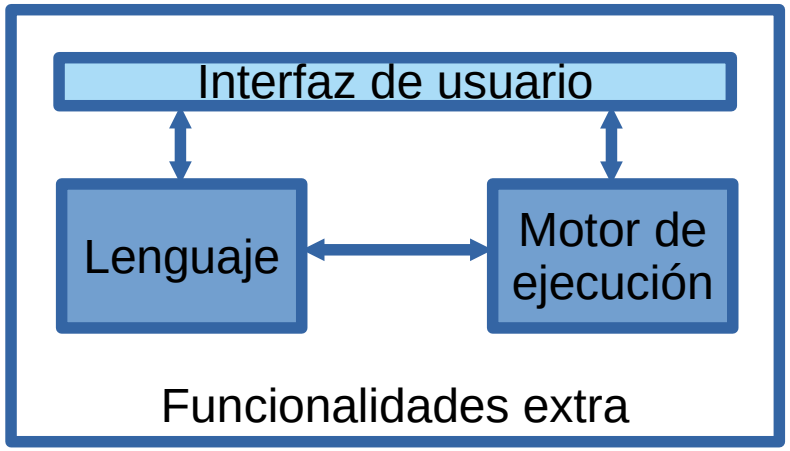
Características de WFS

Gran variedad de WFS, desde más alto a más bajo nivel

- Usuarios expertos o no. Interfaz de usuario
- Aprender un lenguaje de programación
- Independiente o no del código
- Representaciones conceptuales (abstracción) → No contiene información de acceso a los recursos
  * Tratar la complejidad
  * Portabilidad a través de entornos

En general, **WFS se caracterizan:**

1) Cómo describen el WF
2) Cómo ejecuta/maneja los WF
3) Funcionalidades extra

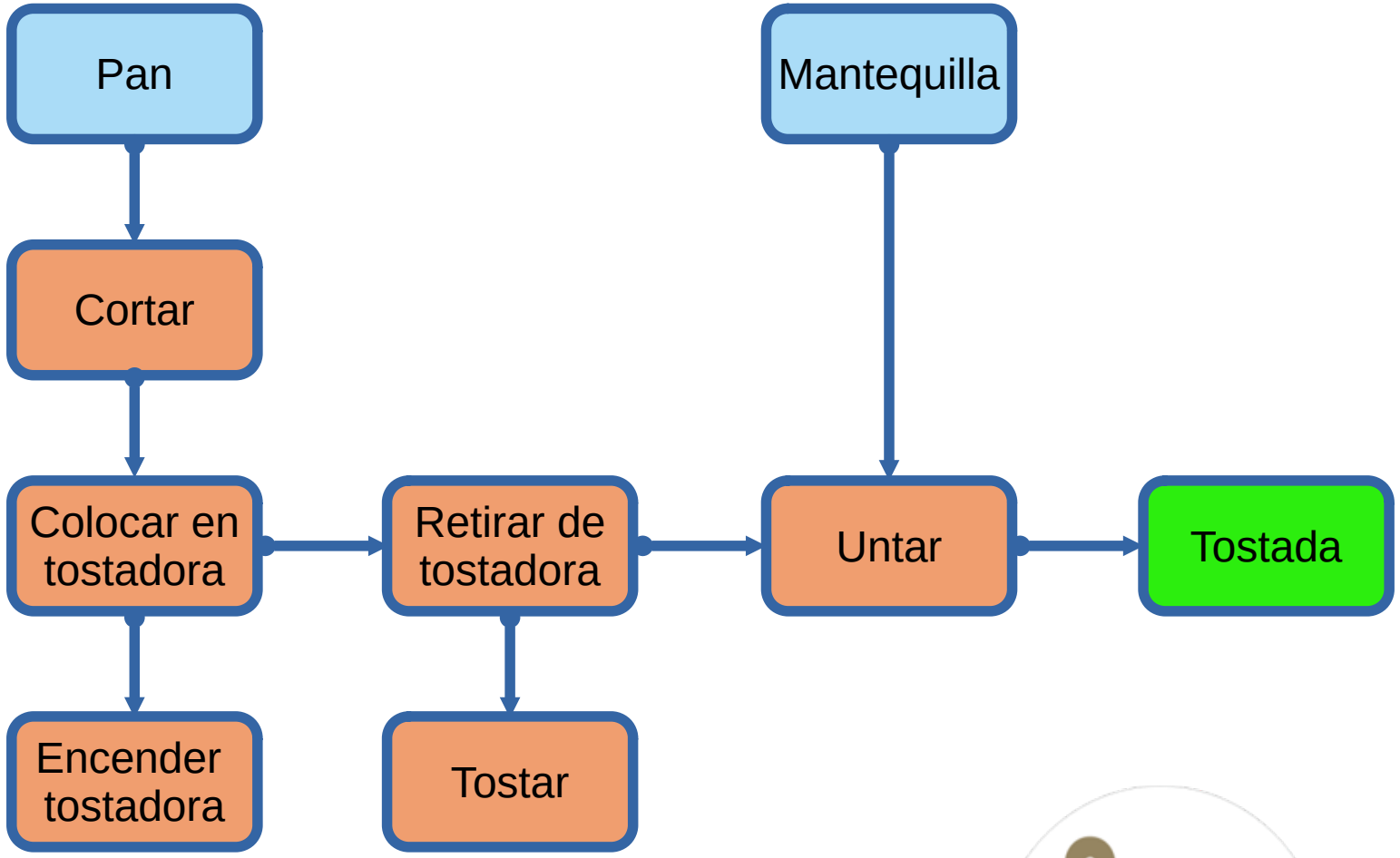


1) Cómo describen el WF

Representaciones textuales

```
Cortar Pan
Colocar_tostadora Pan
Encender_tostadora
Tostar Pan
Retirar_tostadora Pan
Untar Mantequilla
Tostada
```

Interfaz de programación visual



Características de WFS

1) Cómo describen el WF

Representaciones textuales

- Programas con R o Matlab
- Lenguajes de scripts (Python o Perl)
- Recientemente, Jupyter notebooks

Características de WFS

1) Cómo describen el WF

Representaciones textuales

- Programas con R o Matlab
- Lenguajes de scripts (Python o Perl)
- Recientemente, Jupyter notebooks

Interfaz de programación visual

- Gráfico visual conectando nodos
- Traducidos a un lenguaje de especificación de WF (SEL, xSCCULF, MOML, etc.)
- La semántica del grafo puede variar entre herramientas
- Algunos independientes de plataformas
- Adecuados para WF abstractos
- No adecuados para describir en detalle WF extensos y numerosas tareas

Características de WFS

2) Cómo ejecuta/maneja los WF

- Cómo programar las tareas de acuerdo a la lógica
- Manejo de datos
- Monitorización y depuración
- Accesos a datos intermedios
- Aspectos de optimización

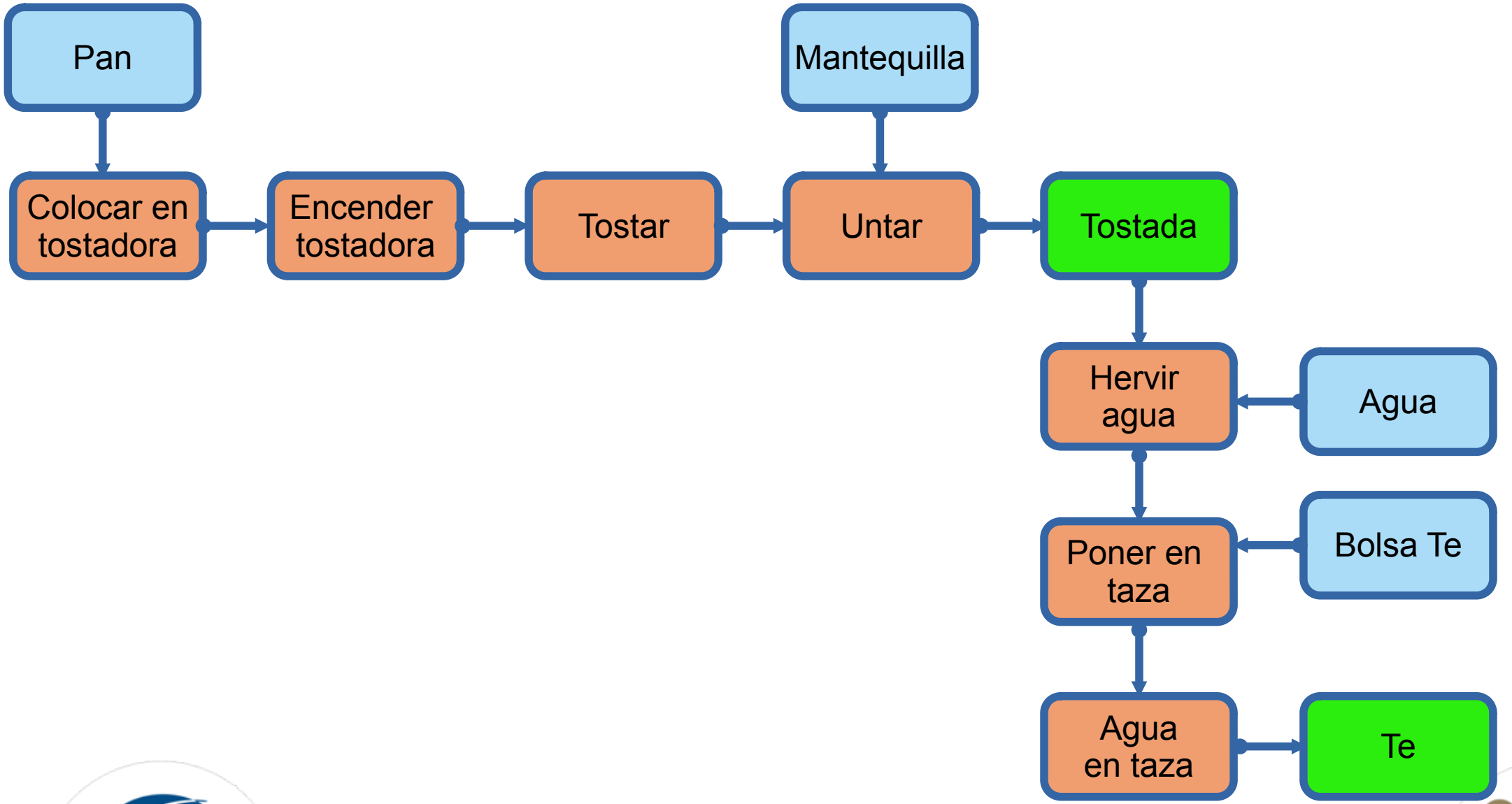
Características de WFS

3) Funcionales extra

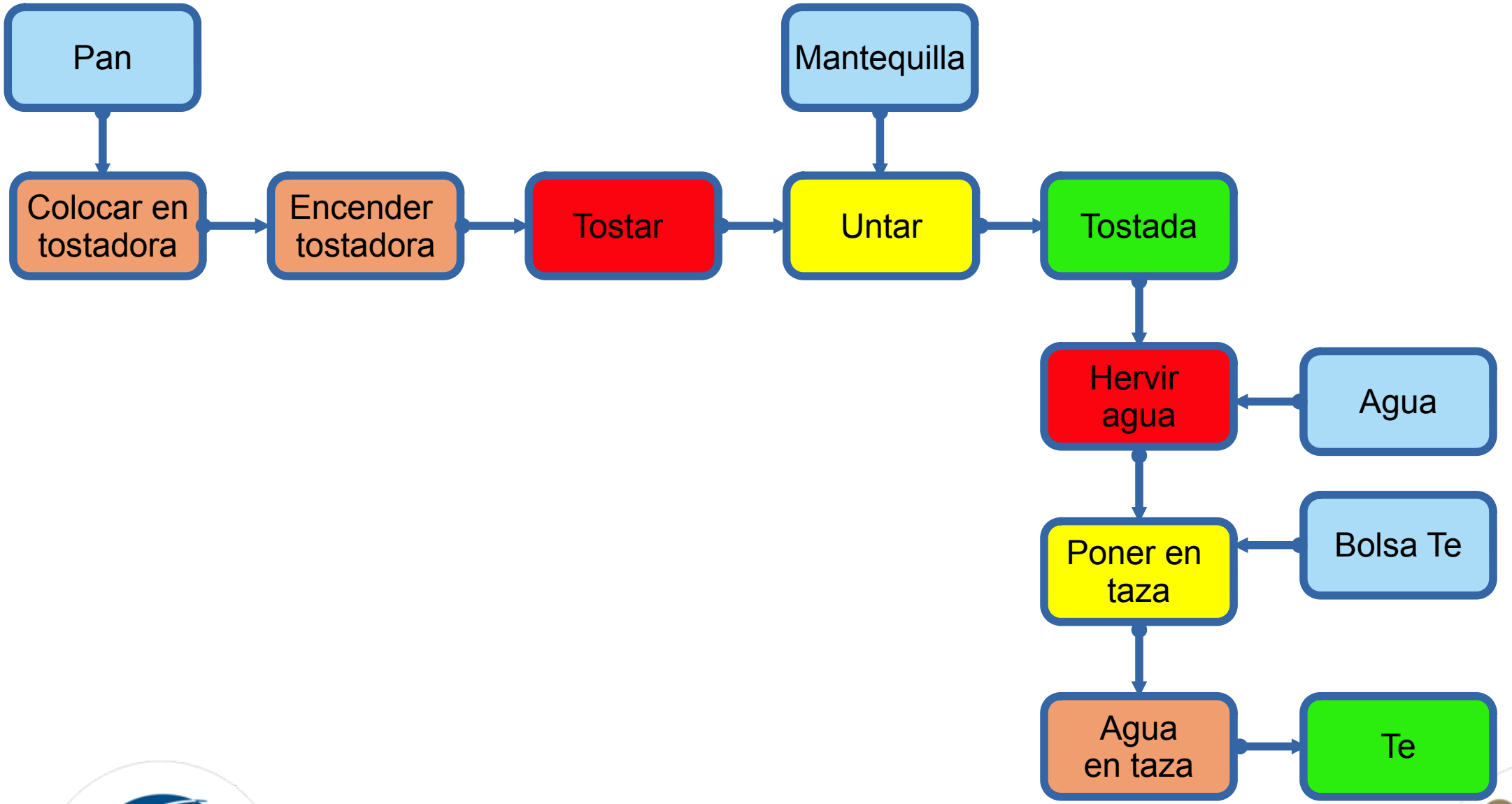
- Asistencia en el diseño y ejecución del WF:
  - Herramientas de diseño intuitivas. Fácilmente personalizables
  - Interfaz de interacción con el usuario
  - Herramientas interactivas para resultados en tiempo real
- Provenance
- Generación de documentación
- Representación visual del WF: **Diagrama de flujo**

  →muy útil para desarrollar y analizar la lógica del WF

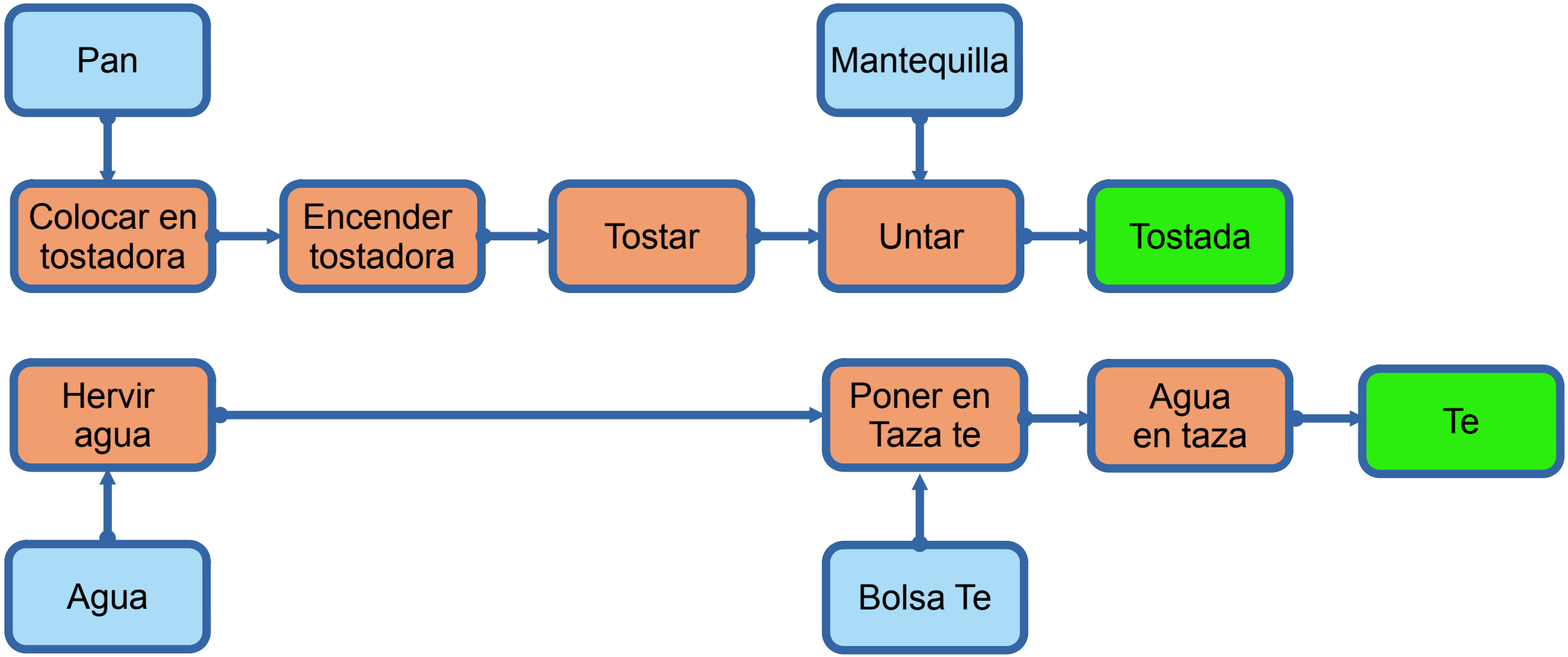
Características de WFS: Diagrama de flujo



Características de WFS: Diagrama de flujo



Características de WFS: Diagrama de flujo



WFS y Proyectos científicos

- Ligeras. No adicional cargas computacionales
- Simples. No conocimiento adicional para implementar y configurar el WF
- Standole. No innecesarias dependencias de otros proyectos y lenguajes de programación
- Activamente soportada
- Open-source y gratuita
- Integración simple con scripts existentes
- Flexibles. Uso de herramientas externas
- Gran volumen de datos y/o recursos computacionales (distribuidos)

  → **WF orientado a Cloud Grid, Cluster**
- Múltiples contribuidores remotos. Ejecución en servidores remotos
- Integración de módulos usando diferentes lenguajes de programación
- Portabilidad (heterogeneidad de plataformas) Transparencia entre sistemas operativos
- Interoperatividad. Actualmente, falta una uniformidad entre herramientas

→ Necesario un estándar del lenguaje de WF

Ejemplos de WFS :

Apache
Taverna

GNU Make

Apache
Airflow

Ejemplos de WFS :

Apache
Taverna

GNU Make

Apache
Airflow

Es una herramienta básica de automatización, bien establecida

GNU Make

- Con mecanismos simples → dependencias complejas
- Asequibles en todos los sistemas operativos
- Capaz de trabajar en paralelo

Es una herramienta básica de automatización, bien establecida

- Con mecanismos simples → dependencias complejas
- Asequibles en todos los sistemas operativos
- Capaz de trabajar en paralelo

WF → reglas: cómo crear las salidas (ficheros de salida)
desde las entradas (ficheros de entrada)

Es una herramienta básica de automatización, bien establecida

GNU Make

- Con mecanismos simples → dependencias complejas
- Asequibles en todos los sistemas operativos
- Capaz de trabajar en paralelo

WF → reglas: cómo crear las salidas (ficheros de salida)
desde las entradas (ficheros de entrada)

```
destino: dependencia1 dependencia2 …...
  (TAB)  orden(es)
```

Es una herramienta básica de automatización, bien establecida

GNU Make

- Con mecanismos simples → dependencias complejas
- Asequibles en todos los sistemas operativos
- Capaz de trabajar en paralelo

WF → reglas: cómo crear las salidas (ficheros de salida)
desde las entradas (ficheros de entrada)

```
destino: dependencia1 dependencia2 …...
  (TAB)  orden(es)
```

```
saludo:
    echo "Buenos días"
```

Make

```
# Fichero: makefile
# Construye un delicioso desayuno

bebida_energetica: azucar bebida0
        poner azucar  in bebida0
        remover azúcar

bebida0: agua bolsa_te
        calentar agua
        poner_taza bolsa _te
        poner_taza agua

tostada: pan mantequilla
poner_tostadora pan
encender_tostadora
untar pan con mantequilla

saludo:
         echo “Buenos días”
```

GNU Make

Make

```
# Fichero: makefile
# Construye un delicioso desayuno

Desayuno: bebida_energetica tostada

bebida_energetica: azucar bebida0
        poner azucar  in bebida0
        remover azucar

bebida0: agua bolsa_te
        calentar agua
        poner_taza bolsa _te
        poner_taza agua

tostada: pan mantequilla
poner_tostadora pan
encender_tostadora
untar pan con mantequilla

saludo:
        echo “Buenos días”
```

destino simbólico

GNU Make

Make

```
# Fichero: makefile
# Construye un delicioso desayuno

Desayuno: bebida_energetica tostada zumo

bebida_energetica: azucar bebida0
	poner azucar  in bebida0
	remover azucar

bebida0: agua bolsa_te
	calentar agua
	poner_taza bolsa _te
	poner_taza agua

tostada: pan mantequilla
	poner_tostadora pan
	encender_tostadora
	untar pan con mantequilla
```

```
zumo: naranjas
	exprimir naranjas
	poner_taza naranjas_exprimidas

saludo:
	echo “Buenos días”
```

GNU Make

Make

```
# Fichero: makefile
# Construye un delicioso desayuno

Desayuno: bebida_energetica tostada zumo

bebida_energetica: azucar bebida0
	poner azucar  in bebida0
	remover azucar

bebida0: agua bolsa_te
	calentar agua
	poner_taza bolsa _te
	poner_taza agua

tostada: pan mantequilla
	poner_tostadora pan
	encender_tostadora
	untar pan con mantequilla
```

```
zumo: naranjas fresas
	exprimir naranjas fresas
	poner_taza frutas_exprimidas

saludo:
	echo “Buenos días”
```

GNU Make

Make: Macros y reglas predefinidas

```
# Fichero: makefile
# Construye un delicioso desayuno

Desayuno: bebida_energetica tostada zumo

bebida_energetica: azucar bebida0
	poner $<  in bebida0
	remover $<

bebida0: agua bolsa_te
	calentar $<
	poner_taza bolsa _te
	poner_taza $<

tostada: pan mantequilla
	poner_tostadora $<
	encender_tostadora
	untar $< con mantequilla
```

```
zumo: naranjas fresas
	exprimir $^
	poner_taza frutas_exprimidas

saludo:
	echo “Buenos días”
```

GNU Make

Macros:

$^ : todas las dependencias de las reglas

$< : Nombre de la primera dependencia de la regla

$@: Nombre del fichero destino de la regla

Make: Macros y reglas predefinidas

```
# Fichero: makefile
# Construye un delicioso desayuno

Desayuno: bebida_energetica tostada zumo

bebida_energetica: azucar bebida0
	poner $<  in bebida0
	remover $<

bebida0: agua bolsa_te
	calentar $<
	poner_taza bolsa _te
	poner_taza $<

tostada: pan mantequilla
	poner_tostadora $<
	encender_tostadora
	untar $< con mantequilla
```

```
zumo: naranjas fresas
	exprimir $^
	poner_taza frutas_exprimidas

saludo:
	echo “Buenos días”
```

GNU Make

Macros:

$^ : todas las dependencias de las reglas

$< : Nombre de la primera dependencia de la regla

$@: Nombre del fichero destino de la regla

Make: Macros

```
# Fichero: makefile
# Construye un delicioso desayuno

LIQUIDO = agua
#(agua, leche)
TIPO = bolsa_te
#(bolsa_te, cafe, colacao)
CREMA = mantequilla
#(mantequilla, tomate, roquefort, …..)
BASE = pan_molde
#(pan_molde, chapata, mollete …..)
ENDULZADOR= azucar
#(azucar, azucar_moreno, sacarina, …..)
FRUTAS = naranjas fresas
#(naranjas, melocotones, piña ...)

Desayuno: bebida_energetica tostada zumo

bebida_energetica: $(ENDULZADOR) bebida0
        poner $<  in bebida0
        remover $<
```

```
bebida0: $(LIQUIDO) $(TIPO)
        calentar $<
        poner_taza $(TIPO)
        poner_taza $<

tostada:  $(BASE) $(CREMA)
        poner_tostadora $<
        encender_tostadora
        untar $< con $(CREMA)

zumo: $(FRUTAS)
        poner_taza frutas_exprimidas

saludo:
        echo “Buenos días”
```

GNU Make

NOMBRE = texto a expandir

Make: Directivas condicionales y loops (for)

```
# Fichero: makefile
# Construye un delicioso desayuno

LIQUIDO = agua
#(agua, leche)
TIPO = bolsa_te
#(bolsa_te, cafe, colacao)
CREMA = mantequilla
#(mantequilla, tomate, roquefort, …..)
BASE = pan_molde
#(pan_molde, chapata, mollete …..)
ENDULZADOR= azucar
#(azucar, azucar_moreno, sacarina, …..)
FRUTAS = naranjas fresas
#(naranjas, melocotones, piña ...)

Desayuno: bebida_energetica tostada zumo
```

```
bebida0: $(LIQUIDO) $(TIPO) $(ENDULZADOR)
	calentar $<
	poner_taza $(TIPO)
	poner_taza $<
	ifdef ENDULZADOR
		poner_taza$(ENDULZADOR)
		remover   $(LIQUIDO)
	endif

tostada: $(BASE) $(CREMA)
	poner_tostadora $<
	encender_tostadora
	untar $< con $(CREMA)

zumo: $(FRUTAS)
	poner_taza frutas_exprimidas

saludo:
	  echo “Buenos días”
```

GNU Make

Make

GNU Make

- **Flexibilidad** → Futuras adaptaciones y extensiones
- **Eficiencia** → Minimiza la carga computacional y el tiempo de ejecución
- **Fiabilidad** → La salida final está actualiza y minimiza los errores

- **Rígido y con funcionalidad limitada**
- **Opaco** ← reglas y macros implícitos

Snake

Combina make & lenguaje Python

→ integra fácilmente en proyectos científicos en Python

Snakefile:

```
rule NAME:
input:
        ”input1”,
        ”input2”,

        ”  ….
output:
        ”output1”,
        ”output2”,
        …..
shell:
        “orden(es) < {input} > “{ouput}
```

Snake

Combina make & lenguaje Python

→ integra fácilmente en proyectos científicos en Python

Snakefile:

```
rule NAME:
input:
        ”input1”,
        ”input2”,

        ” ….
output:
        ”output1”,
        ”output2”,
        …..
shell:
        “orden(es) < {input} > “{ouput}
```

```
def function1(argumens …)
        return out

rule NAME
input:
        ”function1”,
        ”input2”,

        ” ….
output:
        ”output1”,
        ”output2”,
        …..
shell:
        “orden(es) < {input} > {ouput}”
```

Snake

```
rule DESAYUNO:
input:
        ”fruta”,
        ”pan”,
        ”agua”
        ”mantequilla”
         ….
output:
        ”zumo”,
        ”tostada”,
        …..
shell:
        “exprimir < {input} > {ouput}”
        …..
```

Snake

```
ruleorder: DESAYUNO > CAFETERIA

rule DESAYUNO:
input:
        ”fruta”,
        ”pan”,
        ”agua”
        ”mantequilla”
         ….
output:
        ”zumo”,
        ”tostada”,
        …..
shell:
        “exprimir < {input1} > {ouput1}”
        “tostar < {input2} > {ouput2}”
        …..
```

```
Rule CAFETERIA:

input:
        ”cara”,
        ”mesa”,
         ….
output:
        ”zumo”,
        ”tostada”,
        …..
shell:
        “pedir < {input}> {ouput}”
```

Snake

- **Integración con scripts externos**

```
rule DESAYUNO:
input:
        ”fruta”,
        ”pan”,
        ”agua”
        ”mantequilla”
         ….
output:
        ”zumo”,
        ”tostada”,
        …..
script:
        “scripts/tostada.py”
        “scripts/bebida.R”
        “scripts/zumo.jl”
        …..
```

Snake

- **Ejecuta directamente código Python**


```
rule DESAYUNO
input:
        ”naranjas”,
        ”fresas”,
        ”limones”
         ….
output:
        ”zumo”

run:
        for f in input:
                if not f:
                        ….
                        f_exprimir(f)
                with open (output, “w”) as out:
                                out.write(….)
```

Snake

- **Uso de wildcards**


```
rule EXAMPLE:
input:
        "{dataset}/inputfile"
output:
        "{dataset}/file.{group}.txt"
        …..
shell:
        "orden(es) - -group {wildcards.group} <{input} > {ouput}"
```

Snake

- **Dependencia y herencia entre reglas** (usar y combinar WF)

```
rule TOSTAR:
input:
        ”pan”
output:
        a=”pan_tostado”
shell:
        “tostar <{input} > {ouput}”


rule UNTAR:
Input:
        rules.tostar.ouput.a
output:
        pan_untado
shell:
        “untar <{input} > {ouput}”
```

Dependencia

Snake

- **Dependencia y herencia entre reglas** (usar y combinar WF)

```
rule TOSTAR:
input:
        ”pan”
output:
        ”pan_tostado”
shell:
        “tostar <{input} > {ouput}”


use rule TOSTAR as TOSTAR2:
output:
        pan_tostado2
```

Herencia: usa regla existente en un modificado modo

↑

módulos externos

Snake

- Escalables a **servidores, clusters, grids, clouds**

Snake

- Escalables a **servidores, clusters, grids, clouds**
- Fácil manejo de los recursos: **memoria, cores, etc.**


```
rule DESAYUNO:
input:
        ”fruta”,
        ”pan”,
         ….
output:
        ”zumo”,
        ”tostada”,
        …..
threads: 2
resources:
        mem_mb = 100
        nvidia_gpu =2
shell:
        ...
```

Snake

- Escalables a **servidores, clusters, grids, clouds**
- Fácil manejo de los recursos: **memoria, cores, etc.**
- Integración con **Jupyter notebooks**


```
rule DESAYUNO:
input:
        ”fruta”,
        ”pan”,
        ”agua”
        ”mantequilla”
         ….
output:
        ”zumo”,
        ”tostada”,
        …..
notebook:
        “notebool/desayuno.py.ipynb”
```

Snake

- Escalables a **servidores, clusters, grids, clouds**
- Fácil manejo de los recursos: **memoria, cores, etc.**
- Integración con **Jupyter notebooks**
- Manejo de versiones de software y sus dependencias: **Conda y/o contenedores**

```
rule DESAYUNO:
input:
        ”fruta”,
        ”pan”,
         ….
output:
        ”zumo”,
        ”tostada”,
        …..
conda:
        “envs/desayuno.yaml”
container:
        “docker://containers/some_recipe#2.3.2”
shell:
        ...
```

envs/desayuno.yaml

```
channels
        - conda_forge
        - bioconda
        - defaults

dependencias:
        software_cafetera=modelo1.21
        software_tostadora=modelo=1.12
```

Snake

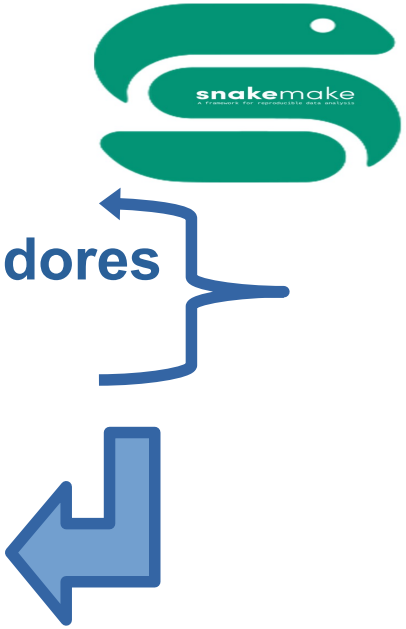
- Escalables a **servidores, clusters, grids, clouds**
- Fácil manejo de los recursos: **memoria, cores, etc.**
- Integración con **Jupyter notebooks**
- Manejo de versiones de software y sus dependencias: **Conda y/o contenedores**
- Crea Ficheros de **Logs** y mensajes por consola

```
rule DESAYUNO:
input:
        ”fruta”,
        ”pan”,
         ….
output:
        ”zumo”,
        ”tostada”,
        …..
message:
        “vamos a preparar un rico desayuno”
log:
        “out=logs/desayuno.out”
        “err=logs/desayuno.out”
shell:
        ...
```

Snake

- Escalables a **servidores, clusters, grids, clouds**
- Fácil manejo de los recursos: **memoria, core, etc.**
- Integración con **Jupyter notebooks**
- Manejo de versiones de software y sus dependencias: **Conda y/o contenedores**
- Crea Ficheros de **Logs** y mensajes por consola

```
rule DESAYUNO:
input:
        ”fruta”,
        ”pan”,
         ….
output:
        ”zumo”,
        ”tostada”,
        …..
message:
        “vamos a preparar un rico desayuno”
log:
        “out=logs/desayuno.out”
        “err=logs/desayuno.out”
shell:
        ...
```



Reproducibilidad

Provenance


```
├── .gitignore
├── README.md
├── LICENSE.md
├── config
│   └── config.yaml
├── workflow
│   ├── envs
│   │   ├── env1.yaml
│   │   └── env2.yaml
│   ├── report
│   │   ├── fig1.rst
│   │   ├── fig2.rst
│   │   └── workflow.rst
│   ├── rules
│   │   ├── rules1.smk
│   │   └── rules2.smk
│   ├── scripts
│   │   ├── script1.py
│   │   └── script2.R
│   ├── notebooks
│   │   ├── notebook1.py.ipynb
│   │   └── notebook2.r.ipynb
│   └── Snakefile
├── resources
└── results
```

WF y Buenas prácticas

WF → principios FAIR (localizable, accesible, interoperable, reutiliazable)

- Almacenar el flujo en repositorios **(github)**
- Updownload a Zenodo y adquirir **DOI**
- Uso de **control de versiones** → archivar las principales versiones con su identificador persistente (DOI)
- Diseño (Plalnificación):
  - **Modularizar**
    - **Descomponer** las tareas en subtareas más pequeñas
    - Identificar componentes **genéricos** y extraerlos del WF
    - Escribir reglas más genéricas ➔ dedicado repositorio
  - Uso de comodines
  - **Explícito,** WF más compresible (para nosotros y otros)
  - Evitar complejas dependencias.
- Documentación de software
- licencia accesibles
- Software open-source
- User-Friendly Poco conocimiento a priori
- Desarrollo de software ágil. Temprano resultados. Feedback

Referencias

- https://github-wiki-see.page/m/common-workflow-language/common-workflow-language/wiki/Existing-Workflow-systems
- GNU Make Manual," Free Software Foundation. Last updated Oct 5, 2014
- https://snakemake.readthedocs.io/en/stable/snakefiles/rules.html
- T. Fernando,et al. "WorkflowDSL: Scalable Workflow Execution with Provenance for Data Analysis Applications," *COMPSAC* 2018, pp. 774-779, doi: 10.1109/COMPSAC.2018.00115

Gracias !!

I hope this is the beginning of a beautiful friendship

with WFS!!!